Bounded Delay and Concurrency for Earliest Query Answering

Olivier Gauwin[1 2 3], Joachim Niehren[1 3], and Sophie Tison[2 3]

[1] INRIA, Lille
[2] University of Lille 1
[3] Mostrare project, INRIA & LIFL (CNRS UMR8022)

Abstract. Earliest query answering is needed for streaming XML processing with optimal memory management. We study the feasibility of earliest query answering for node selection queries. Tractable queries are distinguished by a bounded number of concurrently alive answer candidates at every time point, and a bounded delay for node selection. We show that both properties are decidable in polynomial time for queries defined by deterministic automata for unranked trees. Our results are obtained by reduction to the bounded valuedness problem for recognizable relations between unranked trees.

1 Introduction

Streaming algorithms are relevant for XML databases and data exchange, whenever large data collections are to be processed that cannot be stored in main memory. Instead data is communicated over streams and processed incrementally. Recently, XML streaming algorithms were proposed for schema validation [1] (membership in tree languages), one-pass typing [2] (annotating nodes of trees by types), and query answering [3–5].

The space complexity of streaming algorithms for answering node selection queries in XML trees depends on the size of the call stack (bounded by the depth of the tree) and on the number of concurrently alive answer candidates that are kept in main memory at every time point [6]. The purpose of earliest query answering (EQA) is to minimize the second number. Selection and unselection of answer candidates needs to be decided as early as possible, so that selected candidates can be output and unselected candidates discarded as early as possible. In both cases they are removed from main memory.

EQA is the objective of various streaming algorithms for Forward XPath and its fragments [6–8], and has been studied for automata defined queries too [9, 10]. In the latter paper, it is shown how to obtain a correct EQA algorithm for a fragment of Forward XPath under schema assumptions, by a P-time translation to deterministic streaming tree automata (dSTAs) [11], or equivalently deterministic nested word [12], visibly pushdown [13] or pushdown forest automata [14].

Whether EQA is tractable depends on two properties of the considered query Q and schema S, both of which are independent of the concrete algorithm. The

first restriction is *bounded delay of selection*, which requires a bound for all trees $t \in S$ on the number of events between the first visit of a selected node $\pi \in Q(t)$ and the earliest event that permits its selection. This limits the waiting time for the answer. The second restriction is *bounded concurrency of alive candidates*, which imposes a bound on the number of concurrently alive answer candidates for Q wrt. S for all trees of S at all time points. This limits the maximal number of candidates that need to be memoized simultaneously by every EQA algorithm.

In this paper, we show that bounded delay and bounded concurrency are decidable in P-time for queries and schemas defined by deterministic automata for unranked trees. Our result holds for dSTAs, as well as for bottom-up deterministic tree automata that operate on binary encodings of unranked trees [15], either firstchild-nextsibling based or Curried [16]. When restricting databases to words instead of trees, decision procedures for bounded delay and concurrency for queries defined by dFAs can be obtained quite easily by reduction to bounded ambiguity of nFAs. The algorithm for words, however, cannot be lifted to trees in any straightforward manner. In order to solve this problem, we propose another solution by reduction to the bounded valuedness problem of recognizable relations between unranked trees. As we show, this problem can be reduced to bounded valuedness of bottom-up tree transducers, which can be decided in P-time (Theorem 2.8 of [17]). All reductions are in P-time since we start from deterministic automata. *Omitted proofs can be found in the long version.*

2 Queries in Words

We recall definitions of schemas and queries for tuples of positions in words by dFAs, and the concept of bounded ambiguity for nFAs.

Words, Positions, and Events. Let an alphabet Σ be a finite set with elements ranged over by a, b, c and $\mathbb{N}$ the set of natural numbers $n \geq 1$. We will consider words $w \in \Sigma^+$ as databases. The set of all words Σ^* is closed under concatenation ww' and contains the empty word ϵ. The set of positions of a word $w \in \Sigma^*$ is $pos(w) = \{1, \ldots, |w|\}$ where $|w|$ is the number of letters of w. For all $w \in \Sigma^+$ and positions $\pi \in pos(w)$ we define $lab^w(\pi) \in \Sigma$ to be the π-th letter of word w and say that position π is labeled by $lab^w(\pi)$.

One-way finite automata process words letter by letter from the left to the right, equally to streaming algorithms for words. We define the set of events for a word $w \in \Sigma^+$ by adding the start event 0 to the set of all positions $eve(w) = \{0\} \cup pos(w)$. For all events $e \in eve(w)$, we define $w^{\leq e} \in \Sigma^*$ to be the prefix of w with exactly e letters. We say that two words coincide until event e if $w^{\leq e} = w'^{\leq e}$ and write $eq_e(w, w')$ in this case.

Queries and Schemas. An *n-ary query* Q selects a set of n-tuples of positions for every word $w \in \Sigma^+$. It is a function which maps words w to sets of tuples of positions $Q(w) \subseteq pos(w)^n$. A *schema* $S \subseteq \Sigma^+$ restricts the set of permitted databases. We say that a word $w \in \Sigma^+$ satisfies a schema S if and only if $w \in S$.

Automata, Ambiguity, and Determinism. A *finite automaton* (nFA) over Σ is a tuple $A = (stat, init, rul, fin)$ where $init, fin \subseteq stat$ are finite sets and

$rul \subseteq stat^2 \times (\Sigma \cup \{\epsilon\})$ contains rules that we write as $q \xrightarrow{a} q'$ or $q \xrightarrow{\epsilon} q'$ where $q, q' \in stat$ and $a \in \Sigma$. Whenever necessary, we will index the components of A by A. Let the size of A count all states and rules, *i.e.*, $|A| = |stat_A| + |rul_A|$.

A run of A on a word w is a function $r : eve(w) \to stat_A$ so that $r(0) \in init_A$ and $r(\pi{-}1) \xrightarrow{\epsilon}{}^{*} \xrightarrow{a} \xrightarrow{\epsilon}{}^{*} r(\pi)$ is justified by rul for all $\pi \in pos(w)$ with $a = lab^w(\pi)$. A run is successful if $r(|w|) \in fin_A$. The language $L(A) \subseteq \Sigma^*$ is the set of all words that permit a successful run by A. An nFA is called productive, if all its states are used in some successful run. This is the case if all states are reachable from some initial state, and if for all states, some final state can be reached.

The *ambiguity* $amb_A(w)$ is the number of successful runs of A on w. The ambiguity of A is *k-bounded* if $amb_A(w) \leq k$ for all $w \in \Sigma^*$. It is *bounded*, if it is bounded by some k. An nFA is *deterministic* or a dFA if it has at most one initial state and at most one rule for all left hand sides, including letters. Clearly the ambiguity of dFAs is 1-bounded.

Stearns and Hunt [18] present a P-time algorithm for deciding k-bounded ambiguity of nFAs. Let us write $p \xrightarrow{w} q$ *by* A if there exists a run of $A[init = \{p\}]$ (the automaton obtained from A by setting its initial states to $\{p\}$) on w that ends in q. Weber and Seidl [19] show that an nFA A has unbounded ambiguity iff there exists a word $w \in \Sigma^+$ and distinct states $p \neq q$ such that $p \xrightarrow{w} p$, $p \xrightarrow{w} q$, $q \xrightarrow{w} q$ by A. This can be tested in $O(|A|^3)$ as shown very recently by [20].

Canonicity and Types. Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans. For words $w \in \Sigma^*$ and tuples $\tau = (\pi_1, \ldots, \pi_n) \in pos(w)^n$, let $w * \tau$ be the annotated word in $(\Sigma \times \mathbb{B}^n)^*$ obtained from w by relabeling all positions $\pi \in pos(w)$ to $(lab^w(\pi), b_1, \ldots, b_n)$, where $b_i = 1$ if $\pi = \pi_i$ and $b_i = 0$ otherwise. The *canonical language of an n-ary query Q* is $can_Q = \{w * \tau \mid \tau \in Q(w)\}$. The type of a word $v \in (\mathbb{B}^n)^*$ is an element of $(\mathbb{N} \cup \{0\})^n$ defined by $\sum_{\pi \in pos(v)} lab^v(\pi)$. The type of a word in $(\Sigma \times \mathbb{B}^n)^*$ is the type of its projection in $(\mathbb{B}^n)^*$. All words over $\Sigma \times \mathbb{B}^n$ of type 1^n have the form $w * \tau$, and vice versa. An nFA A over $\Sigma \times \mathbb{B}^n$ is called *canonical* if all words of $L(A)$ have type 1^n. For productive canonical dFAs A, every state $q \in stat$ has a unique type in $\mathbb{B}^n$, which is the type of all words with runs by A ending in q (e.g., Lemma 3 of [21]).

3 Earliest Query Answering for Words

We recall the framework of EQA for XML databases from [10], but restricted to the case of words, and show for queries defined by dFAs that bounded concurrency and delay can be reduced in P-time to bounded ambiguity for dFAs.

An EQA algorithm decides selection and failure of answer candidates at every time point (without knowing the rest of the stream). This way, it needs to keep in main memory only alive candidates, which are neither safe for selection nor failure. As an example, consider the monadic query Q that selects all positions in words w that are labeled by a and followed by bb. When applied to $w = aabbabbcabab$, this query returns $Q(w) = \{(2), (5)\}$. A streaming algorithm can enumerate these answers by using a sliding window of length 3. Position 1 for

instance can be refused when having seen the labels of positions $1, 2$, while position 2 can be selected when having seen the labels of positions $2, 3, 4$.

Schema assumptions are relevant to EQA since restricting the remainder of the stream. The schema $(a|b)^*c(ab)^*$, for instance, excludes all positions from $Q(w)$ that are on the right of the c letter in w. This allows to exclude positions $8, \ldots, 12$ to belong to the answer set $Q(w)$ immediately at the respective position.

Earliest Selection and Bounded Delay. The delay of a selected position is the number of subsequent events before selection can be safely decided. More formally, let Q be an n-ary query in words $w \in \Sigma^+$ satisfying a schema $S \subseteq \Sigma^+$. We define a relation $sel^S_Q(w)$ that links tuples $\tau \in pos(w)^n$ to events $e \in eve(w)$ that are sufficient for selection, *i.e.*, where τ will be selected in all possible continuations of the stream beyond e. Only those continuations are allowed, which extend the current prefix of the word to a member of S:

$$(\tau, e) \in sel^S_Q(w) \;\Leftrightarrow\; \tau \in \{1, \ldots, e\}^n \wedge \forall w' \in S.\ eq_e(w, w') \Rightarrow \tau \in Q(w')$$

Note that the initial event 0 may be sufficient to select the empty tuple () in Boolean queries where $n = 0$, while it is never sufficient for selection if $n \geq 1$ since otherwise $\tau \notin \{1, \ldots, e\}^n$.

Let $latest((\pi_1, \ldots, \pi_n)) = max_{1 \leq i \leq n} \pi_i$ be the latest position of the tuple. The delay of an n-ary query Q for a tuple $\tau \in pos(w)$ is the number of events e following $latest(\tau)$ such that e is insufficient for selection, *i.e.*, $(\tau, e) \notin sel^S_Q(w)$.

$$delay^S_Q(w, \tau) = |\{e \in eve(w) \mid\ latest(\tau) \leq e,\ (\tau, e) \notin sel^S_Q(w)\}|$$

Query Q with schema S has k-bounded delay if $delay^S_Q(w, \tau) \leq k$ for all $w \in S$ and $\tau \in Q(w)$. It has bounded delay if it has k-bounded delay for some $k \geq 0$. Having bounded delay means that every EQA algorithm will output selected tuples a constant time after completion.

Deciding Bounded Delay. We start with the case without schemas. Let A be a canonical and productive dFA over $\Sigma \times \mathbb{B}^n$ and Q_A the n-ary query that it defines. We call a state $q \in stat_A$ of type 1^n *safe for selection* if $(\Sigma \times \{0\}^n)^* \subseteq L(A[init = \{q\}])$. Since A is canonical and deterministic, this is the case if and only if all states reachable from q are final and have outgoing transitions for all letters in $\Sigma \times \{0\}^n$. Thus, the set of states of A that are safe for selection can be computed in time $O(|A| + |\Sigma| + n)$.

Lemma 1. *If the run r of a canonical dFA A on $w * \tau$ exists then it satisfies for all $e \in pos(w)$ that $r(e)$ is safe for selection if and only if $(\tau, e) \in sel_{Q_A}(w)$.*

We define an nFA $D(A)$ such that $amb_{D(A)}(w * \tau) = delay_{Q_A}(w, \tau)$ for all $\tau \in Q_A(w)$. $D(A)$ has the same states as A except for one additional state ok, which is the only final state of $D(A)$. All transitions of A are preserved by $D(A)$. In addition to simulating A, automaton $D(A)$ has ϵ transitions into the state ok from all states q of type 1^n of A that are unsafe for selection. State ok has transitions into itself for all letters in $\Sigma \times \{0\}^n$.

Proposition 1. *For $w \in \Sigma^*$, $\tau \in Q_A(w)$: $delay_{Q_A}(w, \tau) = amb_{D(A)}(w * \tau)$.*

Proof. Consider a run of $D(A)$ on a canonical word $w * \tau$. The only ambiguity of $D(A)$ is introduced by the ϵ-transitions, by which to exit from A at positions between the last component of τ (included) and the earliest event that is safe for the selection of τ. The number of such positions is precisely $delay_{Q_A}(w, \tau)$.

Theorem 1. *Bounded delay for queries Q_A and schemas $L(B)$ defined by dFAs A, B can be decided in time $O(|A| \cdot |B|)$, and k-bounded delay in P-time.*

Proof. We sketch the proof for bounded delay without schemas, where $L(B) = \Sigma^*$. By Proposition 1, it is sufficient to decide whether $D(A)$ has bounded ambiguity. By Weber and Seidl's characterization, this holds if the subautomaton of A containing only unsafe states for selection of type 1^n, has no loop. Acyclicity of this subautomaton can be tested in time $O(|A|)$.

Earliest Failure and Bounded Concurrency. The space complexity of EQA algorithms depends on the concurrency of a query, which is the maximal number of concurrently alive answer candidates at every time point [6, 7], since these are to be kept in main memory. In order to formalize this for n-ary queries, we have to deal with *partial* answer candidates for a given word w. We fix a constant $\bullet$ that represents unknown components, and define partial tuples τ of positions until $e \in pos(w)$ as members of $(\{1, \ldots, e\} \uplus \{\bullet\})^n$. So far, we have only studied *complete* answer candidates, which do not contain any unknown component. We write $compl(\tau, w, e)$ for the set of complete candidates, in which all unknown components of τ have been instantiated with nodes $\pi \in pos(w)$ such that $e \leq \pi$. Given a query Q, schema S, and word $w \in S$, we call a partial candidate τ *failed at event* $e \in eve(w)$, if no completion of τ by nodes in the future of e is selected by Q.

$$(\tau, e) \in fail_Q^S(w) \Leftrightarrow \begin{cases} \tau \in (\{1, \ldots, e\} \uplus \{\bullet\})^n \wedge \\ \forall w' \in S.\ eq_e(w, w') \Rightarrow \forall \tau' \in compl(\tau, w', e).\ \tau' \notin Q(w') \end{cases}$$

A partial candidate $\tau \in (\{1, \ldots, e\} \cup \{\bullet\})^n$ is *alive* at e if it is neither failed nor selected at e. The concurrency of a query schema pair on a word $w \in \Sigma^+$ at position $e \in eve(w)$ is the number of alive candidates at time point e, so that e is neither sufficient for selection or failure:

$$(\tau, e) \in alive_Q^S(w) \Leftrightarrow\ (\tau, e) \notin fail_Q^S(w) \text{ and } (\tau, e) \notin sel_Q^S(w)$$
$$concur_Q^S(w, e) = |\{\tau \in (\{1, \ldots, e\} \cup \{\bullet\})^n \ \mid\ (\tau, e) \in alive_Q^S(w)\}|$$

We say that the concurrency of a query schema pair is bounded if there exists $k \geq 0$ such that $concur_Q^S(w, e) \leq k$ for all words $w \in S$ and $e \in pos(w)$.

Deciding Bounded Concurrency. We start with queries without schemas. So let A be a canonical productive dFA over $\Sigma \times \mathbb{B}^n$ and Q_A the query it defines. Recall that all states of A have a unique type $v \in \mathbb{B}^n$. We call a state q of type v *safe for failure*, if no final states can be reached from q by words of complementary type $1^n - v$. By canonicity, this is the case if no final states can be reached from q at all. We can thus compute the set of safe states for failure in time $O(|A|)$.

Lemma 2. *If the unique run r of a canonical dFA A on some $w * \tau$ exists, then all $e \in pos(w)$ satisfy that $r(e)$ is safe for failure if and only if $(\tau, e) \in fail_{Q_A}(w)$.*

We define an nFA $C(A)$ such that $amb_{C(A)}(w * e) = concur_Q(w, e)$ for all $e \in pos(w)$. The situation is a little different than for $D(A)$ since $C(A)$ runs on words annotated by events rather than tuples. So the alphabet of $C(A)$ is $\Sigma \times \mathbb{B}$. The nFA $C(A)$ guesses all partial candidates with positions until e, tests whether they are alive at e, and accepts in this case and only this case.

Proposition 2. *For all $e \in pos(w)$: $concur_{Q_A}(w, e) = amb_{C(A)}(w * e)$.*

Theorem 2. *Bounded and k-bounded concurrency for queries and schemas defined by canonical dFAs can be decided in P-time for any fixed $k \geq 0$.*

4 Earliest Query Answering for Unranked Trees

We extend EQA from words to unranked trees. We then lift our P-time decision results to tree automata for unranked trees, and argue why the proofs for words cannot be lifted in any straightforward manner.

The set of *unranked trees* T_Σ is the least set that contains all (k+1)-tuples $a(t_1, \ldots, t_k)$ where $k \geq 0$, $a \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma$. Positions of words correspond to nodes of trees, defined by $nod(a(t_1, \ldots, t_k)) = \{\epsilon\} \cup \{i\pi \mid \pi \in nod(t_i)\}$. The word $w = abaca$, for instance, can be encoded by the tree $t = r(a, b, a, c, a)$, where $r \in \Sigma$ is an arbitrary symbol. Note that $nod(t) = \{\epsilon\} \cup pos(w)$. Queries Q in unranked trees select tuples of nodes $Q(t) \subseteq nod(t)^n$ for all trees $t \in T_\Sigma$. Events are produced by preorder traversals:

$$eve(t) = \{\texttt{start}\} \cup (\{\texttt{open}, \texttt{close}\} \times nod(t))$$

There is an initial event `start` and an opening and a closing event per node. Let $\leq$ be the total order on $eve(t)$ induced by preorder traversals over trees $t \in T_\Sigma$, and let $\texttt{pred}(e)$ be the immediate predecessor of event $e \in eve(t) - \{\texttt{start}\}$. For all $e \in eve(t) - \{\texttt{start}\}$, we define the prefix $t^{\leq e}$ of t to be the tree which contains the part of t with all nodes opened before e, *i.e.*, $nod(t^{\leq e}) = \{\pi \in nod(t) \mid (\texttt{open}, \pi) \leq e\}$, and $lab^{t^{\leq e}}(\pi) = lab^t(\pi)$ for all $\pi \in nod(t^{\leq e})$. Note that $t^{\leq(\texttt{close},\pi)}$ contains all proper descendants of π in t, while $t^{\leq(\texttt{open},\pi)}$ does not. As before, we can define $eq_e(t, t')$ by $e = \texttt{start}$ or $t^{\leq e} = t'^{\leq e}$. The notion $t * \tau \in T_{\Sigma \times \mathbb{B}^n}$ extends straightforwardly from words to trees. The canonical language of an n-ary query Q thus has type $can_Q \subseteq T_{\Sigma \times \mathbb{B}^n}$. The definitions of sel^S_Q, $fail^S_Q$, $delay^S_Q$ and $concur^S_Q$ extend literally, except that the set $\{1, \ldots, e\}$ needs to be replaced by $nod(t^{\leq e})$.

Tree automata for unranked trees are often obtained from standard tree automata for binary trees. A binary signature is a finite set $\Gamma = \Gamma_2 \uplus \Gamma_0$ with constants in Γ_0 and binary function symbols in Γ_2. The set of binary trees T^{bin}_Γ is the least set containing all $c \in \Gamma_0$ and triples $f(t_1, t_2)$ where $f \in \Gamma_2$ and $t_1, t_2 \in T^{\text{bin}}_\Gamma$. A tree automaton (TA) for binary trees in T^{bin}_Γ is a tuple $A = (stat, fin, rul)$

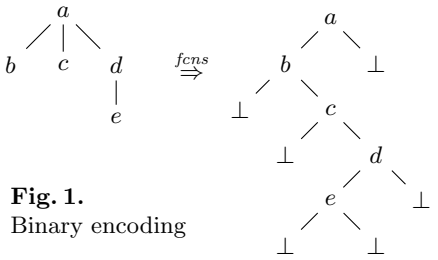


Fig. 1. Binary encoding

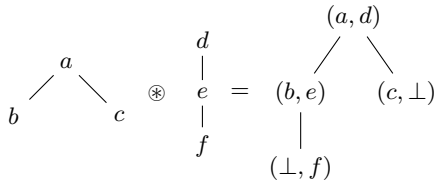


Fig. 2. Example for overlays

consisting of finite sets $fin \subseteq stat$ and a set $rul \subseteq \cup_{i\in\{0,2\}} stat^{i+1} \times \Gamma_i$, that we denote as $f(q_1, q_2) \to q$ and $c \to q$ where $q_1, q_2, q \in stat$, $f \in \Gamma_2$ and $c \in \Gamma_0$. A run of A on $t \in T_\Gamma^{\text{bin}}$ is a function $r : nod(t) \to stat$ such that $f(r(\pi 1), r(\pi 2)) \to r(\pi)$ belongs to rul_A for all nodes π of t with $lab^t(\pi) = f \in \Gamma_2$, and $r(\pi) \to c$ in rul_A for all nodes π of t with $lab^t(\pi) = c \in \Gamma_0$. The language $L^{\text{bin}}(A)$ is the set of all binary trees over Γ that permit an successful run by A, where $r(\epsilon) \in fin$. A (bottom-up) deterministic TA (dTA) is a TA of which no two rules have the same left hand side.

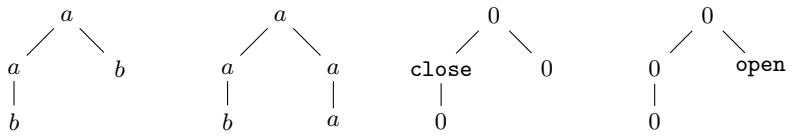
We can encode unranked trees $t \in T_\Sigma$ into binary trees by applying Rabin's firstchild-nextsibling encoding $fcns : T_\Sigma \to T_{\Sigma_\perp}^{\text{bin}}$ where $\Sigma_\perp = \Sigma \uplus \{\perp\}$. The definition is recalled by example in Fig. 1. A TA over $T_{\Sigma_\perp}$ defines the language of unranked trees $L(A) = \{t \in T_\Sigma \mid fcns(t) \in L^{\text{bin}}(A)\}$.

Operationally, however, dTAs fail to operate in streaming manner on unranked trees, so that the previous decision algorithms cannot be lifted to queries defined by dTAs. Streaming tree automata (STAs) [11] operate in the proper order. They are a reformulation of nested word automata [12, 13] and shown equivalent to pushdown forest automata [14]. Deterministic STAs (dSTAs) can perform one-pass typing for extended DTDs with restrained competition [2] as well as EQA [10] for queries defined by dSTAs. Furthermore, deterministic stepwise tree automata [16] can be converted in dSTAs in linear time.

Proposition 3 (Closure properties). *The classes of TAs (wrt. the fcns encoding) and STAs permit determinization, and recognize the same langages of unranked trees modulo P-time automata translations (not preserving determinism). Recognizable languages are closed under Boolean operations, projection and cylindrification. All corresponding operations on TAs (resp. STAs) can be performed in P-time and preserve determinism except for projection.*

Even with STAs, it remains difficult to lift our P-time algorithms for words to trees, since the notion of safe states becomes more complex. Given a canonical dSTA A for query Q_A, one can define another dSTA $E(A)$ for which appropriate notions of safe states wrt. Q_A exist [10]. The size of $E(A)$, however, may grow exponentially in $|A|$. Therefore, we cannot use $E(A)$ to construct polynomially sized counterparts of $D(A)$ and $C(A)$ in the case of unranked trees. Nevertheless:

Theorem 3 (Main). *Bounded delay is decidable in P-time for n-ary queries and schemas in unranked trees defined by dTAs (wrt. the fcns or Curried en-*



(a) Tree t (b) Tree s (c) Tree s_e (d) Tree $s_{e'}$

Fig. 3. $(t, s, s_e) \in Eq$ but $(t, s, s_{e'}) \notin Eq$

coding) or dSTAs, where n may be variable. Bounded concurrency is decidable in P-time for fixed n. For fixed k and n, k-bounded delay and concurrency are decidable in NP-time.

Our proof will be based on the powerful notion of recognizable relations between unranked trees (see [15] for ranked trees). Bounded delay and concurrency are reduced to bounded valuedness of recognizable relations, which in turn is reduced to bounded valuedness of tree transducers for binary trees [17].

5 Recognizable Relations between Unranked Trees

We extend the theory of recognizable relations from ranked to unranked trees. We show that FO-formulas over recognizable relations with n free variables define recognizable relations between n unranked trees (so that satisfiability is decidable), and that bounded valuedness of recognizable relations can be decided in P-time by reduction to bounded valuedness of tree transducers (for binary trees).

Recognizable Relations. In this section, we assume an arbitrary class of tree automata, that satisfy the properties of STAs in Proposition 3. This includes TAs modulo the *fcns* encoding, STAs, and stepwise tree automata [16] (but not deterministic hedge automata with dFAs for horizontal languages [15]).

The *overlay* of k unranked trees $t_i \in T_{\Sigma^i}$ is the unranked tree $t_1 \circledast \ldots \circledast t_k$ in $T_{\Sigma_\perp^1 \times \ldots \times \Sigma_\perp^k}$ obtained by superposing these k trees top-down and left-to-right; the $\perp$ symbol represent missing children where the structures of the trees differ. This is illustrated in Fig. 2. Overlays of ranked trees can be obtained this way too [15], except that overlayed symbols need to inherit the maximal arity. A k-ary relation R between unranked trees is *recognizable* iff the language of its overlays $ovl(R) = \{t_1 \circledast \ldots \circledast t_k \mid (t_1, \ldots, t_k) \in R\}$ is recognizable by a tree automaton. We say that R is recognized by the automaton A if $ovl(R) = L(A)$. We also say that R can be computed in time k if an automaton recognizing R can be computed in time k.

The prime example is the relation $Eq \subseteq T_\Sigma \times T_\Sigma \times T_{\{0,\texttt{open},\texttt{close}\}}$. Here, we map event $e = (\alpha, \pi) \in eve(t)$ to trees $ren^e(t) \in T_{\{0,\texttt{open},\texttt{close}\}}$ obtained by relabeling t, such that π is relabeled to α and all other nodes to 0. We then define $Eq = \{(t, s, ren^e(t)) \mid eq_e(t, s)\}$. See Fig. 3 for an example.

Lemma 3. *Given a signature Σ, a deterministic automaton recognizing relation $Eq \subseteq T_\Sigma \times T_\Sigma \times T_{\{0,open,close\}}$ can be computed in time $O(|\Sigma|^2)$.*

An STA recognizing $ovl(Eq)$ with $O(1)$ states is easy to define. It can be converted into a TA by Proposition 3 and from there to a deterministic automaton of the class under consideration by assumption. The resulting automaton still has $O(1)$ states, and thus an overall size of $O(|\Sigma|)$, if we assume in addition a function ψ such that $|A| \leq |\Sigma| \cdot \psi(|stat_A|)$ for all automata A with signature Σ.

FO Logic. Let Ω be a collection of unranked disjoint signatures and $\Re$ a set of recognizable relations between unranked trees, so that each relation $R \in \Re$ has a type $R \subseteq T_{\Sigma_1^R} \times \ldots \times T_{\Sigma_{ar(R)}^R}$ where $\Sigma_1^R, \ldots, \Sigma_{ar(R)}^R \in \Omega$ and $ar(R) \geq 0$. We fix an infinite set of variables V ranging over unranked trees. A FO formula over recognizable relations in $\Re$ and signatures in Ω has the abstract syntax:

$$\phi \ ::= \ R(X_1, \ldots, X_{ar(R)}) \mid \phi \wedge \phi' \mid \neg\phi \mid \exists X{\in}T_\Sigma.\ \phi$$

where $R \in \Re$, $X_1, \ldots, X_{ar(R)} \in V$, and $\Sigma \in \Omega$. We assume that all formulas are well-typed, *i.e.*, that the types of variables are compatible with those of the relations in which they appear. A formula ϕ with m free variables $X_1, \ldots, X_m$ of types $T_{\Sigma_1}, \ldots, T_{\Sigma_m}$ defines an m-ary relation $R_{\phi(X_1,\ldots,X_m)} \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_m}$. The closure properties of tree automata ensure that all such relations are recognizable. Let $FO_\exists[\Re]$ be the set of existential formulas where existential quantifier are restricted to occur outermost (and Ω is the set of signatures appearing in the types of relations in $\Re$). The size $|\phi|$ is the number of nodes of ϕ.

Proposition 4. *Let $\Re$ be a finite set of relations recognized by deterministic automata $\{A_R\}_{R\in\Re}$. Then there exists a polynomial p such that for all formulas ϕ in $FO_\exists[\Re]$, an automaton recognizing the relation defined by ϕ can be computed in time $p(|\phi|, (|A_R|)_{R\in\Re})$.*

Bounded Valuedness. Let $R \subseteq T_{\Sigma_1} \times T_{\Sigma_2}$ be a recognizable binary relation. For every $s_1 \in T_{\Sigma_1}$, the number $\#R(s_1) = |\{s_2 \mid (s_1, s_2) \in R\}|$ counts the trees in T_{Σ_2} in relation to it. The *valuedness* of R is the maximal such number $val(R) = \max_{s\in T_{\Sigma_1}} \#R(s)$. We call R *bounded* if $val(R) \leq k$ for some $k \geq 0$.

Lemma 4. *A relation R between unranked trees is recognizable iff the corresponding relation between binary trees $fcns(R)$ is, and $val(fcns(R)) = val(R)$.*

Theorem 4. *For every automaton A recognizing a binary relation R between unranked trees, i.e., $L(A) = ovl(R)$:*

1. *$val(R) < \infty$ can be decided in P-time in $|A|$.*
2. *$val(R) < k$ (for a fixed k) can be decided in NP-time in $|A|$.*

Proof. It is sufficient to prove the theorem for TAs over binary trees by Lemma 4. We start with recognizable relabeling relations, and lift the result to recognizable relations in the long version of the paper. A relabeling relation $R \subseteq T_{\Sigma^1} \times \ldots \times T_{\Sigma^n}$ is a relation between trees of the same structure, *i.e.*, whenever $(s_1, \ldots, s_n) \in R$ then $nod(s_1) = \ldots = nod(s_n)$. In other words, the overlays in $ovl(R)$ do not contain any place holder $\perp$.

So let $R \subseteq T_\Sigma \times T_\Delta$ be a relabeling relation for binary signatures, and A a TA for trees in $T_{\Sigma\times\Delta}$ that recognizes R, *i.e.*, $L^{bin}(A) = ovl(R)$. We transform

A into a bottom-up tree transducer T for defining the relation R as in [22]. The rules of T are infered as follows where x_1, x_2 are variables:

$$\frac{(f,g)(p_1,p_2) \to p \in rul(A)}{f(p_1(x_1),p_2(x_2)) \to p(g(x_1,x_2)) \in rul(T)} \qquad \frac{(a,b) \to p \in rul(A)}{a \to p(b) \in rul(T)}$$

This transducer T has the same valuedness as R. Theorem 2.8 of [22] shows that it can be decided in polynomial time whether T is finite-valued, *i.e.*, whether R is bounded. Concerning k-valuedness, it can be decided in non-deterministic polynomial time according to Theorem 2.2 of [22].

The polynomials for testing bounded valuedness of tree transducers are much higher than for testing bounded ambiguity for tree automata [23].

Using the above constructions and Theorem 2.7 of [22], we can build an algorithm for computing the exact value of $val(R)$, if it exists. We can proceed by dichotomy, starting from $2^{2^{P(|A|)}}$, for a fixed polynomial P.

From Proposition 4, we get in P-time a non-deterministic automaton recognizing a relation defined by an $FO_\exists[\Re]$ formula, and then apply Theorem 4:

Corollary 1. *Let $\Re$ be a finite set of relations and A_R deterministic automata recognizing $R \in \Re$. Then there exists a polynomial p such that for formulas ϕ in $FO_\exists[\Re]$, the bounded valuedness $val(R_\phi) < \infty$ of the relation R_ϕ defined by ϕ can be decided in time $p(|\phi|, (|A_R|)_{R\in\Re})$.*

6 Deciding Bounded Delay and Concurrency

We prove the main Theorem 3 on deciding bounded delay and concurrency by reduction to Corollary 1 on recognizable relations.

Let Q be an n-ary query for trees in T_Σ and $S \subseteq T_\Sigma$ a schema. We define a relation $Can_Q = \{(t, ren^\tau(s)) \mid t * \tau \in can_Q \ \wedge \ Eq(t, s, ren^{latest(\tau)}(t))\}$, where $ren^\tau(s)$ is the projection of $s*\tau$ to $\mathbb{B}^n$. The relation $Bef = \{(t, ren^\tau(t), ren^e(t)) \mid \tau \in nod(t^{\leq e})^n\}$ is recognizable by a dTA of size $O(2^n)$, so we cannot use this relation for P-time algorithms without fixing n. By using the relation $Bef\&Can_Q = \{(t, s_\tau, s_e) \mid Can_Q(t, s_\tau)$ and $Bef(t, s_\tau, s_e)\}$, the problem can sometimes be circumvented. Given a deterministic automaton defining Q (it can be a TA on *fcns* encoding, a stepwise tree automaton or an STA), one can construct an automaton of polynomial size recognizing the relation $Bef\&Can_Q$.

Our objective is to define the formulas $delay_Q^S$ and $concur_Q^S$ in the logic $FO_\exists(Eq, Can_Q, S, Bef, Bef\&Can_Q)$, preferably without using Bef. We start with defining relation $Sel_Q^S = \{(t, ren^\tau(t), ren^e(t)) \mid (\tau, e) \in sel_Q^S(t)\}$ by an FO formula $Sel_Q^S(X_t, X_\tau, X_e)$ with three free variables:

$$\begin{aligned} Sel_Q^S(X_t, X_\tau, X_e) \ =_{df} \ & S(X_t) \ \wedge \ Bef(X_t, X_\tau, X_e) \\ & \wedge \forall X'_t \in T_\Sigma.\ (S(X'_t) \ \wedge \ Eq(X_t, X'_t, X_e)) \ \Rightarrow \ Can_Q(X'_t, X_\tau) \end{aligned}$$

Given automata defining Q and schema S, we can thus define an automaton recognizing $Sel_Q^S(X_t, X_\tau, X_e)$. This yields an algorithm for deciding judgments

$(\tau, e) \in sel_Q^S(t)$. It may be unefficient, though, since the automaton obtained this way may be huge, given that formula $Sel_Q^S(X_t, X_\tau, X_e)$ uses full FO-logic of recognizable relations without restriction to some $FO_\exists$.

Bounded Delay. We define the relation $Delay_Q^S = \{(t, ren^\tau(t), ren^e(t)) \mid e \in delay_Q^S(t, \tau)\}$ by the following formula of $FO_\exists(Eq, Bef\&Can_Q, S)$:

$$Delay_Q^S(X_t, X_\tau, X_e) =_{df} \exists X'_t \in T_\Sigma.\ S(X_t) \wedge Bef\&Can_Q(X_t, X_\tau, X_e) \wedge S(X'_t) \wedge Eq(X_t, X'_t, X_e) \wedge \neg Can_Q(X'_t, X_\tau)$$

All base relations can be defined by deterministic automata of polynomial size when leaving n variable (since we don't need relaton *Bef* here). Given deterministic automata A and B defining query Q and schema $S = L(B)$, we can thus define a possibly nondeterministic automaton recognizing $Delay_Q^S(X_t, X_\tau, X_e)$ in P-time from A and B. Let $2Delay_Q^S = \{(t \circledast s_\tau, s_e) \mid Delay_Q^S(t, s_\tau, s_e)\}$. Both relations are recognized by the same automaton. This relation exactly captures the delay: $val(2Delay_Q^S) = |delay_Q^S|$. Applying Corollary 1 to $2Delay_Q^S$ proves that bounded delay is decidable in P-time.

Bounded Concurrency. For concurrency, we proceed in a similar manner. The relation $Alive_Q^S = \{(t, ren^\tau(t), ren^e(t)) \mid \tau \in alive_Q^S(t, e)\}$ can be defined by the following formula of $FO_\exists$:

$$Alive_Q^S(X_t, X_e, X_\tau) = \exists X'_t \in T_\Sigma . \exists X''_t \in T_\Sigma . \exists X'_\tau \in T_{\mathbb{B}^n} . \exists X''_\tau \in T_{\mathbb{B}^n} .\ S(X'_t) \wedge S(X''_t) \wedge Can_Q(X'_t, X'_\tau) \wedge Eq_\Sigma(X_t, X'_t, X_e) \wedge Eq_{\mathbb{B}^n}(X_\tau, X'_\tau, X_e) \wedge Bef_\bullet(X_\tau, X_e) \wedge \neg Can_Q(X''_t, X''_\tau) \wedge Eq_\Sigma(X_t, X''_t, X_e) \wedge Eq_{\mathbb{B}^n}(X_\tau, X''_\tau, X_e) \wedge C_{\mathbb{B}^n}(X''_\tau)$$

Here, $Bef_\bullet$ is like *Bef* but for partial tuples, and $C_{\mathbb{B}^n} \subseteq T_{\mathbb{B}^n}$ is the set of trees of type 1^n. Let $2Alive_Q^S$ be the binary version of $Alive_Q^S$, then $val(2Alive_Q^S) = |concur_Q^S|$. Automata for $C_{\mathbb{B}^n}$ and $Eq_{\mathbb{B}^n}$ are necessarily of size $O(2^n)$, which cannot be avoided by embedding them inside other relation like $\neg Can_Q$. But for a fixed n, all these automata can be computed in P-time, so that Corollary 1 applied to $2Alive_Q^S$ proves that bounded concurrency can be decided in P-time.

Conclusion. In this paper we proved that bounded delay and (for fixed n) bounded concurrency are both computable in P-time, for queries defined by dSTAs. This was obtained by studying some properties of recognizable relations on unranked trees, and combining them with prior results on the valuedness of tree transducers [17]. Considering the P-time translation of a fragment of XPath to dSTAs proposed in [10], we get the same complexity results for this fragment of XPath.

Some questions are left open in the present paper. For fixed k and n, deciding k-boundedness for delay and concurrency for n-ary queries defined by dSTAs is known to be in NP-time. However, NP-hardness is still open. We also chose to define the delay for selection from the time point where the tuple gets complete. An alternative could be to define the ith delay, that starts when i components of the tuple are filled.

Acknowledgments. This work was partially supported by the Enumeration project ANR-07-blanc.

References

1. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: ACM PODS. (2002) 53–64
2. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Transactions of Database Systems **31**(3) (2006) 770–813
3. Gupta, A.K., Suciu, D.: Stream processing of XPath queries with predicates. In: ACM SIGMOD. (2003) 419–430
4. Fernandez, M., Michiels, P., Siméon, J., Stark, M.: XQuery streaming à la carte. In: 23nd International Conference on Data Engineering. (2007) 256–265
5. Benedikt, M., Jeffrey, A.: Efficient and expressive tree filters. In: Foundations of Software Technology and Theoretical Computer Science. LNCS (2007) 461–472
6. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: Buffering in query evaluation over XML streams. In: ACM PODS. (2005) 216–227
7. Olteanu, D.: SPEX: Streamed and progressive evaluation of XPath. IEEE Trans. on Know. Data Eng. **19**(7) (2007) 934–949
8. Gou, G., Chirkova, R.: Efficient algorithms for evaluating XPath over streams. In: ACM SIGMOD. (2007) 269–280
9. Berlea, A.: Online evaluation of regular tree queries. Nordic Journal of Computing **13**(4) (2006) 1–26
10. Gauwin, O., Niehren, J., Tison, S.: Earliest query answering for deterministic streaming tree automata and a fragment of XPath (2009)
11. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Information Processing Letters **109**(1) (2008) 13–17
12. Alur, R.: Marrying words and trees. In: ACM PODS. (2007) 233–242
13. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM Symposium on Theory of Computing. (2004) 202–211
14. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: Foundations of Software Technology and Theoretical Computer Science. (1998) 134–145
15. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M. Available online since 1997: `http://tata.gforge.inria.fr` Revised October, 12th 2007.
16. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: 19th International Conference on Rewriting Techniques and Applications. Volume 3091 of LNCS. (2004) 105–118
17. Seidl, H.: Single-valuedness of tree transducers is decidable in polynomial time. Theoretical Computer Science **106** (1992) 135–181
18. Stearns, R.E., Hunt III, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM Journal on Computing **14**(3) (1985) 598–611
19. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. In: Mathematical Foundations of Computer Science. Volume 233 of LNCS. (1986) 620–629
20. Allauzen, C., Mohri, M., Rastogi, A.: General algorithms for testing the ambiguity of finite automata. In: DLT. Volume 5257 of LNCS. (2008) 108–120
21. Carme, J., Lemay, A., Niehren, J.: Learning node selecting tree transducer from completely annotated examples. In: 7th International Colloquium on Grammatical Inference. Volume 3264 of Lecture Notes in Artificial Intelligence. (2004) 91–102
22. Seidl, H.: Ambiguity, valuedness and costs (1992) Habilitation Thesis.
23. Sakarovitch, J., de Souza, R.: Decidability of bounded valuedness for transducers. In: Mathematical Foundations of Computer Science. (2008)

A Section 3: Earliest Query Answering for Words

Theorem 1 *Bounded delay for queries Q_A and schemas $L(B)$ defined by dFAs A, B can be decided in time $O(|A| \cdot |B|)$, and k-bounded delay in P-time.*

Proof. The case without schemas is proved in the core of the paper. We generalize this proof by analogy, *i.e.*, by introducing an automaton $D(A, B))$ whose ambiguity captures the delay, and then an equivalent criterion on its bounded ambiguity.

If a schema S is provided together with a query Q, a first approach is to define the query Q^S such that $sel^S_Q = sel_{Q^S}$. This can be done with the following definition for Q^S: $Q^S(w) = Q(w)$ if $w \in S$, and $pos(w)^n$ otherwise. However, building an automaton recognizing Q^S cannot be done without a blowup in $O(2^n)$ in the general case.

We propose an adaptation of the construction of $D(A)$ such that it captures the case with schema, in order to avoid this blowup. The automaton $D(A, B)$ operates on the alphabet $\Sigma \times \mathbb{B}^n$. It is somehow a product automaton between A and B, in which we add ϵ transitions on unsafe states. $stat(D(A, B)) = (stat(A) \times stat(B)) \uplus \{ok\}$ with $init(D(A, B)) = init(A) \times init(B)$ and $fin(D(A, B)) = \{ok\}$. Rules are obtained by the following inference schema:

$$\frac{q_A \xrightarrow{(a,b_1,\ldots,b_n)} q'_A \in rul(A) \qquad q_B \xrightarrow{a} q'_B \in rul(B)}{(q_A, q_B) \xrightarrow{(a,b_1,\ldots,b_n)} (q'_A, q'_B) \in rul(D(A, B))}$$

We also add ϵ transitions from safe states to state ok, and rules from state ok to itself for every letter in $\Sigma \times \{0\}^n$. In $D(A, B)$, safe states for selection are states (q, q') of type 1^n such that $L(B[init = q']) \otimes \{0\}^n \subseteq L(A[init = q])$, where $L \otimes \{0\}^n$ are words of $(\Sigma \times \mathbb{B}^n)^*$ of type 0^n with the projection on Σ^* in L. To compute these safe states, we cannot only rely on final states: (q, q') can be safe, with $q \notin fin(A)$ and $q' \notin fin(B)$. However, we can find another property of unsafe states:

1. if there is a word $w \in (\Sigma \times \{0\}^n)^*$ such that one can reach from (q_A, q_B) a state (q'_A, q'_B) in $D(A, B)$ by reading w, with $q'_A \notin fin(A)$ but $q'_B \in fin(B)$ then obviously (q_A, q_B) is unsafe.
2. every state (q_A, q_B) of $D(A, B)$ that can access to a state described above by any word is unsafe.

These two conditions exactly describe unsafe states. Thus safe states can be computed in time $O(|A|.|B|)$ using the following Datalog program:

$$\frac{q_A \xrightarrow{(a,0^n)} q'_A \in rul(A) \qquad q'_A \notin fin(A) \qquad q_B \xrightarrow{a} q'_B \in rul(B) \qquad q'_B \in fin(B)}{(q_A, q_B) \in unsafe_{sel}(D(A, B))}$$

$$\frac{q_A \xrightarrow{(a,b_1,\ldots,b_n)} q'_A \in rul(A) \qquad q_B \xrightarrow{a} q'_B \in rul(B) \qquad (q'_A, q'_B) \in unsafe_{sel}(D(A, B))}{(q_A, q_B) \in unsafe_{sel}(D(A, B))}$$

We get:

$$delay_{Q_A}^{L(B)} = amb(D(A,B))$$

Instead of using a cubic algorithm for deciding the bounded ambiguity of $D(A,B)$ [20], we use an equivalent characterization [19] that we can efficiently adapt here.

Let $D'(A,B)$ be the automaton identical to $D(A,B)$ except that the state ok and its corresponding rules are removed, and $fin(D'(A,B)) = fin(A) \times fin(B)$. We make $D'(A,B)$ productive. Then $delay_{Q_A}^{L(B)} < \infty$ iff the subautomaton of $D'(A,B)$ containing only unsafe states of type 1^n has no loop. Each operation can be done in time $O(|A|.|B|)$.

Theorem 2 *Bounded and k-bounded concurrency for queries and schemas defined by canonical dFAs can be decided P-time for any fixed $k \geq 0$.*

Proof. In the case without schemas, this follows immediately from Proposition 2, the P-time procedure for deciding bounded ambiguity [20], and the fact that $C(A)$ can be constructed in P-time from A. Before constructing $C(A)$, we need to make A productive, which can be done in $O(|A|)$.

Similarly to the delay, we can get rid of the schema by including it to the query, even for *fail*. If Q^S is defined by: $Q^S(w) = Q(w)$ if $w \in S$ and $\emptyset$ otherwise, then $fail_Q^S = fail_{Q^S}$. However, concurrency also depends on sel_Q^S, so for the same reason we would need at least time $O(2^n)$.

But once more we can avoid this blowup. We build $C(A,B)$ from A and B (dFA recognizing S) the same way than for $C(A)$, except that previously we compose rules of A and B as described in proof of Theorem 1. We accept a run for a guessed tuple iff this tuple is alive at the event e given as input. To check this, we need to test if the state of $C(A,B)$ reached at e is neither safe for selection nor safe for failure. We saw in proof of Theorem 1 how to compute safe states for selection. We show here how to compute unsafe states for failure. These are states from which there is a successful run in A and B on a same word. They can be computed in $O(|A|.|B|)$ using the following Datalog program:

$$\frac{q_A \overset{(a,b_1,\ldots,b_n)}{\rightarrow} q'_A \in rul(A) \qquad q'_A \in fin(A) \qquad q_B \overset{a}{\rightarrow} q'_B \in rul(B) \qquad q'_B \in fin(B)}{(q_A,q_B) \in unsafe_{fail}(D(A,B))} \qquad \frac{q_A \overset{(a,b_1,\ldots,b_n)}{\rightarrow} q'_A \in rul(A) \qquad q_B \overset{a}{\rightarrow} q'_B \in rul(B) \qquad (q'_A,q'_B) \in unsafe_{fail}(D(A,B))}{(q_A,q_B) \in unsafe_{fail}(D(A,B))}$$

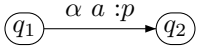
Now the concurrency is exactly the ambiguity of this automaton:

$$concur_{Q_A}^{L(B)} = amb(C(A,B))$$

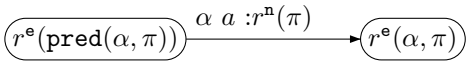
The finite ambiguity of $C(A,B)$ can be decided in P-time, q.e.d.

B Section 5: Recognizable Relations between Unranked Trees

B.1 Streaming Tree Automata

An STA [11] for trees in T_Σ is a tuple $A = (stat, init, fin, rul)$ where $stat = stat^{\mathtt{e}} \uplus stat^{\mathtt{n}}$ is a finite set with two kinds of states (for events and nodes), $init, fin \subseteq stat^{\mathtt{e}}$ and $rul \subseteq stat^2 \times \{\mathtt{open}, \mathtt{close}\} \times \Sigma$, that we denote as 

where $q_1, q_2 \in stat^{\mathtt{e}}$, $\alpha \in \{\mathtt{open}, \mathtt{close}\}$, $p \in stat^{\mathtt{n}}$, and $a \in \Sigma$. A run of an STA on a tree t is a pair of functions $(r^{\mathtt{e}}, r^{\mathtt{n}})$ with types $r^{\mathtt{e}} : eve(t) \to stat^{\mathtt{e}}$ and $r^{\mathtt{n}} : nod(t) \to stat^{\mathtt{n}}$ which map events and nodes to states, such that $r^{\mathtt{e}}(\mathtt{start}) \in init$ and such that the rule



belongs to rul for all $\pi \in nod(t)$ with $a = lab^t(\pi)$, and actions $\alpha \in \{\mathtt{open}, \mathtt{close}\}$. The language $L(A)$ is the set of all unranked trees $t \in T_\Sigma$ that permit a successful run by A, *i.e.*, $r^{\mathtt{e}}((\mathtt{close}, \epsilon)) \in fin$.

An STA is *deterministic* or equivalently a *dSTA*, if it has a single initial state, no two `open` rules for the same letter use the same event state on the left, and no two `close` rules for the same letter use the same node state and the same event state on the left. It should be notice that deterministic stepwise tree automata for unranked trees [16] can be encoded into equivalent dSTAs in linear time.

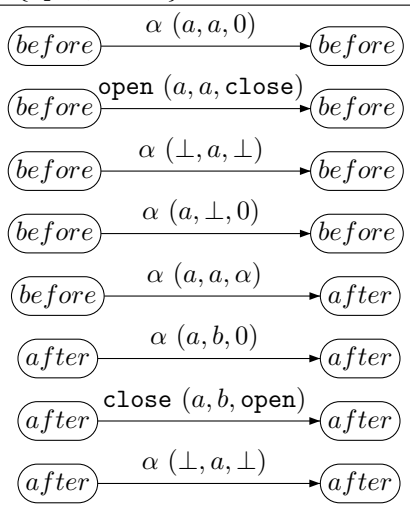
B.2 Recognizability of *Eq*

Lemma 3 *Given a signature Σ, a deterministic automaton recognizing relation $Eq \subseteq T_\Sigma \times T_\Sigma \times T_{\{0, open, close\}}$ can be computed in time $O(|\Sigma|^2)$.*

We define a dSTA A on $\Sigma_\perp \times \Sigma_\perp \times \{0, \mathtt{open}, \mathtt{close}\}_\perp$ such that $L(A) = ovl(Eq)$. We use two event states $stat^{\mathtt{e}}(A) = \{before, after\}$, where $init(A) = \{before\}$ and $fin(A) = \{after\}$. No node states are used. The rules are given by

the following inference schema:

$$\frac{\alpha \in \{\texttt{open}, \texttt{close}\} \qquad a \in \Sigma \qquad b \in \Sigma_\perp}{}$$



B.3 $FO_\exists[\Re]$-definable relations in P-time

Proposition 4 *Let $\Re$ be a finite set of relations recognized by deterministic automata $\{A_R\}_{R\in\Re}$. Then there exists a polynomial p such that for all formulas ϕ in $FO_\exists[\Re]$, an automaton recognizing the relation defined by ϕ can be computed in time $p(|\phi|, (|A_R|)_{R\in\Re})$.*

R is recognizable by Proposition 3 and the standard correspondence between Boolean operation and projection on languages and automata.

In the following, we prove the existence of an automaton A recognizing R, that can be computed in polynomial time.

First we get rid of cartesian products in formulas by introducing existentially quantified fresh variables and adding the relations $\texttt{SameTree}_\Sigma(t, t')$, that just check that $t = t'$ for trees t, t' on the alphabet Σ. For instance, $R(X_1, X_2) =_{df} \exists X_3 \in T_\Sigma.\ R_1(X_1, X_3) \ \wedge \ R_2(X_3, X_2)$ would become: $R(X_1, X_2) =_{df} \exists X_4 \in T_\Sigma.\ \exists X_3 \in T_\Sigma.\ R_1(X_1, X_3) \ \wedge \ R_2(X_4, X_2) \ \wedge \ \texttt{SameTree}_\Sigma(X_3, X_4)$. Obviously, $\texttt{SameTree}_\Sigma$ is recognizable in time $O(|\Sigma|)$, and the size of the formula remains polynomial.

We now prove that Boolean operations can be applied on automata in polynomial time and without losing determinism.

(Negation). Let $R \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_m}$ be a recognizable relation, and A a deterministic automaton with $L(A) = \mathit{ovl}(R)$. A deterministic automaton $\overline{A}$ recognizing $\overline{L(A)}$ can be built in P-time. $\overline{A}$ recognizes the relation $\neg R$:

$$\neg R(s_1, \ldots, s_m) \Leftrightarrow s_1 \circledast \ldots \circledast s_m \notin \mathit{ovl}(R) \Leftrightarrow s_1 \circledast \ldots \circledast s_m \in L(\overline{A})$$

(Conjunction). Let $R_1 \subseteq T_{\Sigma_1^1} \times \ldots \times T_{\Sigma_k^1}$ and $R_2 \subseteq T_{\Sigma_1^2} \times \ldots \times T_{\Sigma_d^2}$ be two recognizable relations, and A_1 and A_2 two deterministic automata with

$L(A_1) = ovl(R_1)$ and $L(A_2) = ovl(R_2)$. Let $R \subseteq T_{\Sigma_1^1} \times \ldots \times T_{\Sigma_k^1} \times T_{\Sigma_1^2} \times \ldots \times T_{\Sigma_d^2}$ be the relation defined by the formula:

$$R(X_1^1, \ldots, X_k^1, X_1^2, \ldots, X_d^2) =_{df} R_1(X_1^1, \ldots, X_k^1) \ \wedge \ R_2(X_1^2, \ldots, X_d^2)$$

Here we supposed, w.l.o.g, that orderings between X_j^i were kept unchanged. By cylindrification, we extend the relations R_1 and R_2 on the signature of R. Let R_1' and R_2' be the relations such that for every trees $(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) \in T_{\Sigma_1^1} \times \ldots \times T_{\Sigma_k^1} \times T_{\Sigma_1^2} \times \ldots \times T_{\Sigma_d^2}$,

$$R_1'(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) \Leftrightarrow R_1(s_1^1, \ldots, s_k^1)$$

and

$$R_2'(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) \Leftrightarrow R_2(s_1^2, \ldots, s_d^2)$$

Automata A_1' and A_2' recognizing R_1' and R_2' can be obtained from A_1 and A_2 by keeping the same states, and adding rules for all the possible completions on the new components, which increases for instance $|A_1|$ by a factor of $|\Sigma_1^1| \ldots |\Sigma_k^1|$.

Now the automaton A recognizing R is just the intersection of A_1' and A_2', which can also be done in polynomial time:

$$\begin{aligned} R(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) &\Leftrightarrow R_1(s_1^1, \ldots, s_k^1) \ \wedge \ R_2(s_1^2, \ldots, s_d^2) \\ &\Leftrightarrow R_1'(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) \ \wedge \ R_2'(s_1^1, \ldots, s_k^1, s_1^2, \ldots, s_d^2) \end{aligned}$$

Note that this holds thanks to the elimination of cartesian products mentioned above, *i.e.*, because $\{X_1^1, \ldots, X_k^1\} \cap \{X_1^2, \ldots, X_d^2\} = \emptyset$.

Hence we have shown that Boolean operations in formulas correspond to Boolean operations on automata recognizing the relation that keep determinism and can be done in P-time. It remains to deal with existential quantification in formulas.

(Existential quantification). Let $R \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_m}$ be the relation defined by the formula:

$$R(X_1, \ldots, X_m) =_{df} \exists X_1' \in T_{\Sigma_1'} \ldots \exists X_k' \in T_{\Sigma_k'}.\ R'(X_1, \ldots, X_m, X_1', \ldots, X_k')$$

If A' is an automaton recognizing R', then, by applying a projection on its last m components, we get an automaton A recognizing R, in polynomial time. This projection does not preserve determinism, this is why quantifiers are required to be outside Boolean connectives. Here we assumed w.l.o.g. that quantified variables appeared as last components of the relation: if not, projection is done on the corresponding components.

B.4 From ranked to unranked trees

To prove Lemma 4, we first prove a correspondence between the overlay of a tree and the overlay of its binary encoding.

Let *ren* be the morphism on binary trees that renames constants $(\bot, \ldots, \bot)$ to $\bot$ and preserves the trees otherwise. This morphism is linear and one-to-one, so it preserves regularity in both directions: L is recognizable iff $ren(L)$ is recognizable. The following lemma relates overlays of unranked and ranked trees.

Lemma 5. $fcns(t_1 \circledast \ldots \circledast t_n) = ren(fcns(t_1) \circledast \ldots \circledast fcns(t_n))$

Hence, notions of recognizablity on R and $fcns(R)$ are equivalent:

Lemma 4 *A relation R between unranked trees is recognizable iff the corresponding relation between binary trees $fcns(R)$ is, and $val(fcns(R)) = val(R)$.*

By definition $fcns(R) = \{(fcns(t_1), \ldots, fcns(t_n)) \mid (t_1, \ldots, t_n) \in R\}$. Lemma 5 yields $ovl(fcns(R)) = ren(fcns(ovl(R)))$. The morphism *ren* preserves recognizability back and forth. Thus, $fcns(R)$ is a recognizable relation iff $ovl(fcns(R))$ is recognizable language of unranked trees iff $fcns(ovl(R))$ is a recognizable language of binary trees iff $ovl(R)$ is a recognizable language of unranked trees iff R is a recognizable relation of unranked trees.

B.5 Deciding valuedness of binary relations

Theorem 4 *For every automaton A recognizing a binary relation R between unranked trees, i.e., $L(A) = ovl(R)$:*

1. *$val(R) < \infty$ can be decided in P-time in $|A|$.*
2. *$val(R) < k$ (for a fixed k) can be decided in NP-time in $|A|$.*

We cannot directly use the transformation used in the preceding lemma as the transducer would not be in the class considered in Theorem 2.2 of [22]. So we slightly modifiy the relation in a relabeling relation while keeping the valuedness. In the same way as before, we just need to consider binary relations over binary trees. So, let R be a recognizable relation over $T_{\Sigma^1} \times T_{\Sigma^2}$ and let us consider that the symbol $(\bot, \bot)$ (resp. $\bot$) can be either of arity 0 or of arity 2.

We associate with R, $C(R)$ the set of trees t in $T_{\Sigma^1_\bot \times \Sigma^2_\bot}$ obtained from trees in $ovl(R)$ by expansion with $(\bot, \bot)$. More precisely, an automaton for $C(R)$ is obtained from the automaton A for $ovl(R)$ just by adding to A one state $q_\bot$ and the following rules:

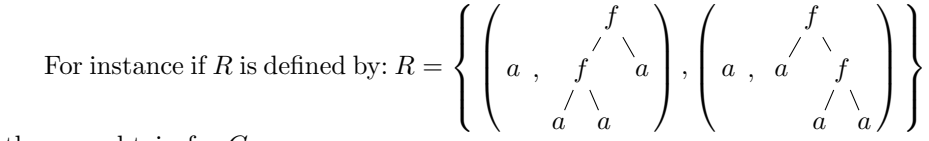
$$(\bot, \bot) \to q_\bot \qquad (\bot, \bot)(q_\bot, q_\bot) \to q_\bot \qquad \frac{\alpha \to q \in rul(A)}{\begin{array}{c}\alpha(q_\bot, q_\bot) \to q \\ \alpha(q_\bot) \to q\end{array}}$$

We denote by $proj_i$ the homomorphism defined by $proj_i(\alpha_1, \alpha_2) = \alpha_i$ for any (α_1, α_2) in $\Sigma^1_\bot \times \Sigma^2_\bot$. $C(R)$ defines a relation C_R over $T_{\Sigma^1_\bot} \times T_{\Sigma^2_\bot}$ by:

$$C_R = \{(proj_1(c), proj_2(c)) \mid c \in C(R)\}$$

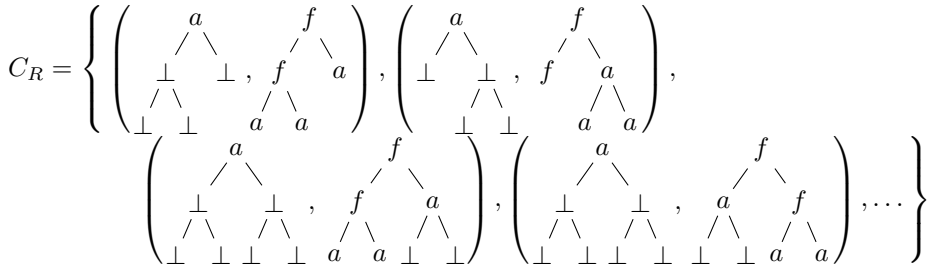
Let us remark that if (c_1, c_2) belongs to C_R, then $nod(c_1) = nod(c_2)$. So $C(R)$ is exactly $ovl(C_R)$.

For instance if R is defined by: $R = \left\{ \left(a\ ,\ f(f(a,a),a) \right), \left(a\ ,\ a(f(a,a)) \right) \right\}$



then we obtain for C_R:

$$C_R = \left\{ \left(f(a,a)\ ,\ f(f(a,a),a) \right), \left(f(a,\bot)\ ,\ f(f(a,a),a) \right), \left(f(\bot,\bot)\ ,\ f(f(a,a),a) \right), \left(f(a,a)\ ,\ f(f(\bot,\bot),a) \right), \left(f(\bot,\bot)\ ,\ f(f(\bot,\bot),a) \right), \left(f(a,a)\ ,\ f(a,f(a,a)) \right), \left(f(\bot,\bot)\ ,\ f(a,f(a,a)) \right), \ldots \right\}$$



and thus we find in C_R infinitely many witnesses of any pair of R.

Now let $clean_i$ the function from $T_{\Sigma^i_\bot}$ to T_{Σ^i} which associates with a term t, the corresponding term without $\bot$, *i.e.*, u such that: $nod(u) = \{\pi \in nod(t) \mid lab^t(\pi) \neq \bot\}$ and $lab^u(\pi) = lab^t(\pi), \forall \pi \in nod(u)$. Furthermore we restrict $clean_i$ to the trees t such that $\{\pi \in nod(t) \mid lab^t(\pi) \neq \bot\}$ is prefix-closed. It is easy to check that a tree c in $T_{\Sigma^1_\bot \times \Sigma^2_\bot}$ belongs to $C(R)$ iff $R(clean_1(proj_1(c)), clean_2(proj_2(c)))$ and that $C_R(u,v)$ iff $R(clean_1(u), clean_2(v))$ and $nod(u) = nod(v)$.

Lemma 6. *C_R and R have the same valuedness.*

First, let us prove that the valuedness of C_R is at least the valuedness of R. Let t in T_{Σ^1} such that there exists at least k distinct t_i with $R(t,t_i)$. Let $D = nod(t) \cup \cup_{i=1}^k nod(t_i)$. For a tree u and a set of nodes D such that $nod(t) \subseteq D$, we define the completion of u w.r.t. D as the tree u^D defined by $nod(u^D) = D$ and $lab^{u^D}(\pi) = lab^u(\pi)$ if p belongs to $nod(u)$, $lab^{u^D}(\pi) = \bot$ otherwise. As $nod(t^D) = nod(t_i^D)$ and $clean_1(t^D) = t, clean_2(t_i^D) = t_i$, we have $C_R(t^D, t_i^D)$, $1 \leq i \leq n$. As the t_i, $1 \leq i \leq n$, are distinct, so are the t_i^D, $1 \leq i \leq n$: the valuedness of C_R is at least the valuedness of R.

Now, let us prove that the valuedness of C_R is at most the valuedness of R. Let u in $T_{\Sigma^1_\bot}$ such that there exists at least k distinct v_i with $C_R(u,v_i)$. Let $t = clean_1(u), t_i = clean_2(v_i)$: we have $R(t,t_i)$. It remains to prove that the t_i are all distinct.

Let $1 \leq i < j \leq n$: as $v_i \neq v_j$ there exists a position π such that $lab^{v_i}(\pi) \neq lab^{v_j}(\pi)$:

- either $lab^{v_i}(\pi) \neq \bot$ and $lab^{v_j}(\pi) \neq \bot$: then π belongs to $nod(t_i)$ and to $nod(t_j)$ and $lab^{t_i}(\pi) \neq lab^{t_j}(\pi)$.
- either $lab^{v_i}(\pi) \neq \bot$ and $lab^{v_j}(\pi) = \bot$: then π belongs to $nod(t_i)$ and π does not belong to $nod(t_j)$.
- either $lab^{v_j}(\pi) \neq \bot$ and $lab^{v_i}(\pi) = \bot$: similar to the precedent case.

So, there exists t in T_{Σ^1} such that there exists at least k distinct t_i with $R(t,t_i)$ q.e.d.

So, computing the valuedness of R can be reduced to computing the valuedness of C_R. Furthermore, as the transformation from an automaton recognizing $ovl(R)$ to an automaton recognizing $ovl(C_R)$ is linear, we get directly the theorem from the proof given in Section 5 for relabelings.